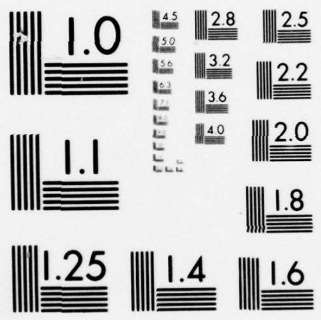
AD-A052 876   NAVAL SURFACE WEAPONS CENTER DAHLGREN LAB VA   F/G 9/2
A COMPARISON OF APPROACHES TO PROVIDE A DEBUGGING SYSTEM FOR TH--ETC(U)
NOV 77   R L HARTUNG

UNCLASSIFIED   NSWC/DL-TR-3729   NL

1 OF 1
AD
A052876

END
DATE
FILMED
5 -78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR-3729	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* (6) A COMPARISON OF APPROACHES TO PROVIDE A DEBUGGING SYSTEM FOR THE AN/UYK-20.		5. TYPE OF REPORT & PERIOD COVERED (9) Final rept.,
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Ronald L. Hartung		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Surface Weapons Center (CK-70) Dahlgren Laboratory Dahlgren, VA 22448		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NIF
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Surface Weapons Center Dahlgren Laboratory Dahlgren, VA 22448		12. REPORT DATE (11) Nov 77 ok per telecon ~~February 1978~~
		13. NUMBER OF PAGES 30 (12) 28 p.
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)* Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)* (14) NSWC/DL-TR-3729		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)* AN/UYK-20 minicomputer Debugging Software approach Virtual machine approach		
20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*		

This study was conducted to establish feasible approaches to debugging programs for the AN/UYK-20, the Navy's standard minicomputer. Five approaches are identified and compared: internal, external, hybrid, software, and virtual machine. The results of this study can be generalized to cover the problems of program debugging for all digital computers.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

391 598

FOREWORD

This report describes a number of approaches for enhancing the Navy standard AN/UYK-20 computer to provide better software debugging capabilities. The advantages and disadvantages of five different approaches were examined. Considering cost, capabilities, advantages, and disadvantages, no single approach seems to stand out as a dominant approach. The approach called external or "black box" was proposed by Mars Gralia and Bill Sanda, both of the Applied Physics Laboratory, Johns Hopkins University. The author wishes to especially acknowledge the efforts of Mars Gralia who provided advice and criticism throughout the study.

This report was prepared in the Programming Systems Branch of the Computer Programming Division, Warfare Analysis Department. It was sponsored by the Naval Electronics Systems Command (NAVELEX), Code 570.

The report was reviewed by H. W. Thombs, Head, Programming System Branch.

Released by:

Ralph A. Niemann

R. A. NIEMANN
Head, Warfare Analysis Department

TABLE OF CONTENTS

LIST OF TABLES

INTRODUCTION

This report contrasts and analyzes five approaches to provide a debugging capability for the AN/UYK-20 minicomputer. The approaches considered are: internal, external, hybrid, software, and virtual machine. The external approach uses a black box plugged into the AN/UYK-20 in place of the existing front panel. The internal approach uses changes in the microprogramming of the AN/UYK-20 to achieve debugging functions. The hybrid approach is a combination of the external and internal approaches, and the software approach refers to conventional software debugging systems. The virtual machine approach requires the construction of a virtual machine monitor within the AN/UYK-20 as well as the running of a software debugging package as a separate virtual machine from the program under test.

To rate the relative merits of the five approaches, the following three questions are considered:

1. What resources are required?

2. What capabilities are provided?

3. What advantages or disadvantages are incurred in versatility, reliability, etc.?

Appendix B mentions possible enhancements to the debugging package that may be desirable.

CONFIGURATION REQUIREMENTS

The configuration requirements for the five debugging approaches are described below. In most cases, adding resources to these configurations will augment the performance of the debugging package. The projections and conclusions of this report are based on the configurations described herein.

The external approach consists of disconnecting the front panel of the AN/UYK-20 and attaching a "black box" which will then assume the function of the front panel. The black box will have an interface for an operator console, probably a teletype or cathode-ray tube (CRT) terminal with an RS-232 interface. In addition, the black box will provide a set of switches and lights which can assume the functions of the standard front panel so that maintenance personnel can run standard diagnostics without learning to operate the CRT-oriented black box. The black box is designed around a microprocessor such as an 8080 or 6800 type and will provide debugging functions without recourse to microcode or mainstore programs in the AN/UYK-20.

The internal approach, on the other hand, uses internal resources of the AN/UYK-20 to provide debugging capabilities. This is done with microcode changes, emulator control word (ECW) changes, and mainstore routines. In addition, one I/O channel is used to communicate with an operator console; this can be any I/O device via any standard interface option of the UYK-20. This device is controlled by a software driver in mainstore. The mainstore program will require 2 to 6K of main memory, depending on the instructions selected. New routines will be added in the micromemory space occupied by the microdiagnostics (address space 3 to 4K of micro store, the fourth micromemory board), and the ECW read only memory (ROM) will also be altered. Thus, in the worst case, two boards must be altered. None of these changes, however, requires hardware changes, merely modification of ROM contents.

The hybrid approach requires all the resources of the pure external approach. It also requires the ECW changes and microcode for the diagnostic board.

The software approach requires 4 to 8K of main memory, depending on the package selected. For this report, the dynamic debug module for the AN/UYK-20 SDEX, specified to have a maximum size of 6000 decimal words, will be used for comparison.

The virtual machine approach* involves changes to the AN/UYK-20 to make it host a virtual machine monitor. Although the AN/UYK-20 presents obstacles to virtualization, it does appear possible. An initial investigation of the possibility of virtualizing AN/UYK-20 computers was conducted by BGS Systems, Incorporated under an NSWC/DL contract dealing with virtual machines and system integration. Because of the possible lack of memory for debugging purposes (some users use all 64K for operational programs), the use of a virtual machine (VM) along with a double density memory has been investigated. With this approach, the user will have a full 64K of core memory for software, and the debugging system will occupy the added memory (up to 64K). The double density memory required for this approach will be developed using semiconductor memory and will use an unused bit in the status register as an additional address bit. This will require an I/O channel to a terminal, some ECW changes, and additional microcode.

CAPABILITIES

To compare the capabilities provided by each debugging approach, a list of possible debugging commands was generated. Although not assumed to be all inclusive or of optimum utility, this list does appear to contain the basic building blocks upon which a wide range of possible debugging commands can be built. For example, the concept of operand fetch or store breakpointing provides sufficient power to provide protection of that operand. Moreover, the fact that a given

* See Appendix C for a more detailed discussion.

method is able to breakpoint on a read or write allows, with simple extensions, the protection of a block of memory. In this sense, the list of debugging commands attempts to convey a sense of completeness in allowing each approach to be rated as to its relative power as a debugging tool. Table 1 displays the list of commands and indicates the ability of each approach to incorporate the commands.

The list of commands is arranged in order of preference for the command, although the rating of these commands was produced with a very limited sample, and the order should be taken as very tentative.* At present, no weighting factor is applied to the commands to indicate their relative value; in some cases, these commands may have evolved from the primitive front panel debugging mode and may not be necessary for an efficient debugging package. The STEP command may fall into this category. Therefore, to better compare the capabilities of the various debugging approaches, a study should be made to determine which commands are most useful. The following discussion of problem areas proceeds in the order of their occurrence on the list.

START AND STOP

Since these two commands are related in function, they are discussed together. The commands allow the user to control the program from outside the program; they do not refer to breakpoints, etc. Although all five debugging approaches allow the user to initiate execution of a module under test, the mechanism for STOP causes problems under some approaches. The external and hybrid approaches stop program execution by use of the hardware stop. The internal, software, and VM approaches, however, cannot use the hardware stop function. Instead, they must either receive an interrupt from an operator terminal or wait for a breakpoint or trap to occur. In the internal and software cases, the module under test can hang in a loop with disabled interrupts, and a STOP command will not be received. In this case, the user may have to manually stop the AN/UYK-20 from the front panel and restart in a debugging mode to examine the current state of the module under test. This problem may be minimized by using time sliced operation for the debugging module which is scheduled by an RTC interrupt, a class II interrupt. By using the class II interrupt level, I/O program interrupts will not hang up the debugger.

The VM approach allows a stop because the debugger is a separate process and cannot be locked out by the program under test.

* An area not covered by this study is the selection of an optimal set of debugging commands.

Table 1. Debugging Commands Implemented by Each Approach

Command	External	Internal	Hybrid Internal/External	Software	Virtual Machine
START	Yes	Yes	Yes	Yes	Yes
STOP	Yes	No	Yes	No	Yes
STEP	Yes	Yes	Yes	Yes	Yes
INSPECT & CHANGE;					
DYNAMIC	No	Yes	Yes	Yes	Yes
STATIC	Yes	Yes	Yes	Yes	Yes
INSTRUCTION BREAKPOINT	Yes	Yes	Yes	Yes	Yes
OPERAND BREAKPOINT	Yes	No	Yes	No	No
JUMP TRACE	No*	Yes	Yes	No*	No*
FORMAT OF BINARY REPRESENTATION	Yes	Yes	Yes	Yes	Yes
RESOLVE ADDRESSES	Yes	Yes	Yes	Yes**	Yes
RESOLVE RELOCATIONS	Yes	Yes	Yes	Yes**	Yes
EVALUATE EXPRESSIONS	Yes	Yes	Yes	Yes**	Yes
DUMP STATIC	Yes	Yes	Yes	Yes	Yes
DUMP SNAP	No	Yes	No	Yes	Yes
BLOCK SET MEMORY	Yes	Yes	Yes	Yes	Yes
SEARCH	Yes	Yes	Yes	Yes	Yes
PATCH AID	Yes	Yes	Yes	Yes**	Yes
CHECKPOINT/ RESTART	Yes	Yes	Yes	Yes**	Yes
SYMBOLIC DEBUG	Yes	Yes	Yes	Yes**	Yes

* Breakpoints can be used, but a large number of breakpoints may be required.
** Not implemented in dynamic debug for SDEX.
Requires external mass storage.
Requires new version assembler or compiler plus additional memory for symbolic table.

INSPECT AND CHANGE

Two types of INSPECT AND CHANGE command are considered: static and dynamic. Static INSPECT AND CHANGE is the capability to examine and change the registers or memory of the AN/UYK-20 while it is halted. This capability includes access to mainstore and all registers which are program controllable. All five approaches provide this capability. Dynamic INSPECT AND CHANGE is the ability to inspect and change memory or registers while the AN/UYK-20 under test is running. In a tactical environment, this is accomplished by time slicing the debugging module. The software, internal, and VM approaches have this capability. The straight hardware approach must stop the AN/UYK-20 CPU in order to gain access, which could have a very adverse effect on interrupts, particularly if multiple locations or registers are to be accessed. The only hindrance to the hybrid approach is the fact that it is slowed down by the response time of the cable to the front panel and the speed of the microprocessor in the black box. However, this approach appears to be feasible overall.

INSTRUCTION BREAKPOINT

All five approaches are capable of the INSTRUCTION BREAKPOINT command. The internal, hybrid, and VM approaches can implement the breakpoint sequence in microcode, thus allowing faster response and less interference with program execution. This can be done using the diagnostic jump instruction with no changes to the ECW ROM.

The suggestion from some programmers of a breakpoint covering access to an area of memory is not possible with the AN/UYK-20 without simulation in some sense. The VM approach might do this by trapping in 1K pages and then checking for restricted accesses.

OPERAND BREAKPOINT

The construction of the OPERAND BREAKPOINT command allows the debugging module to monitor operands. The hardware of the front panel allows one such breakpoint to be set which will trap on the READ or WRITE mode, or both of these. Thus, the external and hybrid approaches allow the operand breakpoint. This function is not available to the internal, software, or VM approaches, and they cannot breakpoint on an operand fetch or store.

JUMP TRACE

This command allows the debugging package to maintain a table of the last m jumps, where m is determined by the size of the table. The capability can be achieved by breakpointing all jumps, but this could require an indefinitely large jump table. The internal, hybrid, and VM approaches are capable of adding this feature without the use of a breakpoint table by modification of the ECW ROM and added microcode. This approach is faster than the breakpoint performed with an external box or with software.

DUMP SNAP

The DUMP SNAP command, activated by breakpoints or timer interrupts, displays selected portions of mainstore and registers. This function should be accomplished quickly without stopping the CPU so that interrupts may still be handled. The external and hybrid approaches, however, require stopping the CPU and sending out data over the front panel cables; this could cause interrupt problems due to the expected slower I/O rate over the cables to the front panel.

OTHER COMMANDS

There are few differences in the capabilities of each approach, and commands on the list which show no differences are not discussed. Appendix A contains a discussion of all the commands listed.

POSSIBLE FUTURE EXPANSIONS

The capabilities discussed cover the area of debugging. However, it may be desirable to expand the debugging facility to include or aid in other functions such as performance monitoring, validation, and verification. The internal and VM approaches are more suitable for assisting in these areas. (See Appendix B for more details.)

Finally, two observations can be made on the subject of capabilities:

1. The external approach does not add any functions over the normal front panel functions, but does automate them and adapt them for operator use.

2. The hybrid approach adds a dynamic INSPECT AND CHANGE capability to the external approach, but may still be limited by I/O rate over the cable.

A possible course would be to extend the external approach by microstepping the AN/UYK-20 and monitoring its operation via the micro P, source bus, and microinstruction. This means the box would stop the CPU, thus creating potential interrupt problems. It appears that the processor could achieve one-half speed at best in this mode.

ADVANTAGES AND DISADVANTAGES

This section discusses for each approach those issues not related to debugging capabilities; Table 2 summarizes the discussion. The table is formulated around the idea that an advantage for one approach is considered a disadvantage for the other by its absence.

Table 2. Advantages and Disadvantages of Debugging Approaches

		Advantages		
External	Internal	Hybrid	Software	Virtual Machine
Protected from user intervention	Runs faster than other methods	Protected from user intervention	No changes to AN/UYK-20 hardware or firmware	Requires one I/O channel
Requires no machine resources	Can run time sliced w/o stopping processor	Requires no mainstore or I/O channel	Can run time sliced	No interference with program address space
Can be turned on & off externally	Faster I/O rate than external or hybrid	Can run time sliced but may still have interrupt problems	Easy to take to the field	Run time sliced
	Easy to take to the field	Can use microcode speed up external approach		Protected from user
				Can make use of microcode to speed up operation
				Easy to take to the field
		Disadvantages		
Requires new hardware	Uses some mainstore & one I/O channel	Requires hardware & microcode development	Uses core and an I/O channel	Requires development of VM and a double density memory
Packaging problems	Diagnostic board is removed	Packaging problems		Uses one I/O channel
	Requires microcode development			Requires microcode development

EXTERNAL APPROACH

The external approach is appealing in that it requires no machine resources, does not require internal modification of the AN/UYK-20, and can be turned on or off. Its effect on operating speed can be removed when debugging is not needed. Its main disadvantages are the requirement for new hardware to be developed, maintained, and supported; creation of packaging problems; and slowdown in the operation of the AN/UYK-20 under some modes.

The slower mode of operation is caused by a 1- to 2-μs cycle microprocessor in the black box; thus, a data item cannot be processed faster than several microseconds. In addition, the bus to the front panel is not an I/O port and does not have a stable signal, so that the data must be sampled to determine if they are valid, costing several microprocessor cycles. Since the Univac drawings contain no part numbers, it was not possible to calculate speed of transmission; it is unlikely, however, that the bus was designed for high speeds since it is used to drive LED's for display. This is significant in cases like snap dumps or dynamic inspect and change where the CPU is to be stopped for a minimum time period.

The area of packaging is noncritical in laboratory environments. However, it is doubtful that a programmer will want to leave behind his debugging tool when the program goes into the fleet. NSWC/DL has found that when program problems appear aboard ship, the debugging must often be performed on the ship; therefore, the black box must be able to be taken aboard ship. There are two ways to package the black box, each with its own problems. First, the box can be a direct replacement for the front panel. In this case, the panel will be oversize since additional hardware will be included. The second packaging method involves opening the front panel and cabling to the front panel connectors which may cause EMI/RFI problems because of exposure of the cable to the ship's environment. In either method, the connectors to the front panel appear rather fragile and could result in connector failure problems.

INTERNAL APPROACH

The internal approach has several advantages. Since it uses microcode, it can implement some operations faster than a software or hardware approach. In particular, the breakpoint OP code performs most of the breakpoint functions and runs in microcode allowing better dynamic inspect and change and snap dumps. In addition, the debugging package can be scheduled by an executive to run time sliced with the module under test, facilitating inspect and change in a dynamic sense. The internal approach also uses the AN/UYK-20 I/O channel, thus requiring less CPU time for I/O than the external or hybrid approaches.

The main disadvantages of the internal approach are the uses of mainstore, an I/O channel, and microcode. It is estimated that this approach requires 4 to 6K of mainstore. In addition, since the diagnostic board is removed, the micro-diagnostics are less convenient because boards have to be swapped. Also, the user's program can interfere with the code of the debugger because it resides in the same physical memory.

HYBRID APPROACH

The hybrid approach attempts to take advantage of both the internal and external approaches. It uses no core or I/O channels, but does require changes to the microcode and ECW ROM. It is somewhat faster than the external approach and can run the front panel from the microcode in a more desirable manner. It can also run time sliced and is fully protected from the user. However, as shown later, it is more costly than the internal and external approaches because hardware and microcode must both be developed. The hybrid approach suffers from the same physical disadvantages as the external approach.

SOFTWARE APPROACH

The software approach is the conventional approach. It has the advantage of already existing in unofficial forms as well as in an official version which has been specified for use with SDX. However, it does consume mainstore and an I/O channel; software debuggers typically require 4 to 8K of core memory. Although they use CPU time, software debuggers can be scheduled for minimum impact. Furthermore, they are are field deployable.

VIRTUAL MACHINE APPROACH

The virtual machine (VM) approach requires one I/O channel and some microcode changes (ECW and diagnostic board). It also requires the double density memory board. It is the most expensive of the approaches, but also the most flexible. The debugging package can be made by modifying an existing AN/UYK-20 package to allow communication between the debugging VM and the operational program under test. In addition to the debugging applications, the VM and the extended AN/UYK-20 memory may be useful in other areas.

COSTS

Table 3 gives projected cost information on the various debugging approaches.

Table 3. Cost Projections for Debugging Approaches*

Approach	Cost ($)
EXTERNAL	
Development	75,000
Production units	2,000 ea.
INTERNAL	
Development	75,000
Production units	800
HYBRID	
Development	80,000 - 90,000
Production	2,800 ea.
VM	
Development of VM system	100,000
Double density board	50,000
Production	1,000 - 2,000
Double density board	4,000

NOTE: The cost of reverifying the AN/UYK-20 is not included in these costs.
* The cost of the software debugging approach is not shown since there are existing software debugging packages in use or under development.

Developmental costs for both internal and external approaches seem to be equal. That is, the cost of designing the new hardware in the external approach offsets the microcode modifications to the AN/UYK-20 in the internal approach, while the effort involved in producing the debugging mainstore code in the internal approach is approximately equal to the effort involved in programming the microprocessor in the external approach. Developmental cost is estimated on the basis of two persons for six to nine man-months. Hardware costs, as compared with labor costs, are insignificant in both approaches; thus, $75,000* is required to develop either system. The production cost of the external approach is approximately $2K per unit in 300-unit quantities. The internal approach requires two special AN/UYK-20 boards which should cost about the same as the standard boards since only ROM contents are changed; thus, at an estimated $400 per board, the cost is $800 in production quantities.

The developmental costs of the hybrid approach exceed the pure approaches by $5K to $15K. Since the hybrid approach requires the two special boards plus the external box, the expected production cost is $2.8K per unit.

The VM approach requires two developments: (1) the VM modification of the AN/UYK-20 microcode and (2) the development of the double density memory board. In each case, a full two-month evaluation must be performed as the first phase of the development. The production cost of the VM approach is based on two to four boards, depending on the changes required. Since the memory board does not have to be MIL-SPEC, the cost is considerably lower. In addition, an AN/UYK-20 debugging program has to be modified to work with the VM. This should cost less than $1000.

SUMMARY

The following discussion shows that the five debugging approaches, internal, external, hybrid, virtual machine (VM), and software, are relatively equal in capability. The cost for development of the internal and external approaches is about the same, but the external approach costs a little more in production. The hybrid approach is not a great improvement over the other approaches; it appears to provide little gain for its increased cost. The VM approach is interesting in that it provides not only a debugging package, but the capabilities of virtual machines on the AN/UYK-20 and a double density memory. The software approach is a low-risk, low-cost approach. Table 4 shows the overall results of this study.

* Does not include selecting/determining which debugging functions to implement.

Table 4. Comparison of Debugging Approaches

	Software	External	Internal	Hybrid	Virtual Machine
Development cost	Low	Medium	Medium	Medium	High
Risk area	None	Hardware	Microcode	Hardware & Microcode	Hardware & Microcode
Production cost	Low	Medium	Low	Medium	High
Portability*	High	Medium	High	Medium	Medium
Hardware changes	None	Door	Two cards	Door**	Memory**
Machine resources	High	Low	High	Low	Medium

* Terminal required; may be on-site.
** Plus change two cards.

APPENDIX A

EXPLANATION OF COMMAND TYPES

This appendix details the intended operation of the command types. This document does not attempt to give these commands appropriate or final form for use in a well-designed debugging system. Instead, they are intended as basic building blocks required for a debugging system.

START, STOP, AND STEP

This set of commands is basic to the operation of any debugging package. The functions of START and STOP are obvious as the user must be able to initiate and halt operation of the software to be debugged. In particular, the STOP command must be able to cause a halt in case of endless loops. The STEP operation is a carryover from the early front panel debugging methods. It may be useful as both a high-level debugging aid and a machine language instruction step.

INSPECT AND CHANGE

The scope of this command covers a very wide range. First of all, it applies to all registers (general, status, program counter, etc.) and to mainstore. It allows the user to modify the states of the AN/UYK-20 and is used to patch code. However, it includes another dimension, static or dynamic. The static INSPECT AND CHANGE refers to its use when the machine is not running the software under test. This is the normal case in most debugging packages. The dynamic case, however, allows the monitoring and modifying of a program and data while the program is running. This is done in some existing AN/UYK-20 debugging packages by allowing a monitor to run along with the program in a time shared mode. This also provides the facility to do a SNAP DUMP at intervals or perhaps when triggered by a BREAKPOINT.

BREAKPOINT

This command includes two subtypes, operand and instruction. An INSTRUCTION BREAKPOINT allows the debugging package to halt program execution and perform some debugging function either by operator intervention or by a previously specified command (e.g., SNAP DUMP). The OPERAND BREAKPOINT traps on the condition that a read and/or write occurs to a given location in memory. In addition to the above, a BREAKPOINT can be masked so that a range of addresses will trip the BREAKPOINT.

JUMP TRACE

This command traces the last n branches or jumps taken in the program to provide a trace of how the program progressed from its starting to present location.

FORMAT OF BINARY REPRESENTATION

This is not so much a command by itself, but a part of all other commands. It is necessary to at least present data in octal format, but the presentation of data in HEX, ASCII, instruction mnemonics, decimal, and floating point could be added.

RESOLVE ADDRESSES

This command allows the user to find the effective address of an operand when using direct, indirect, or indexed addressing.

RESOLVE RELOCATIONS

This command is used to find a particular location in a module which has been relocated. This assumes a table of starting addresses for the modules and that the user gives the displacement into the module of interest.

EVALUATE EXPRESSIONS

This command allows the user to calculate results to compare with results of programmed execution, e.g., What is 45 octal + E9 HEX in octal?

FORMATTED DUMP

This command produces a dump, in octal or HEX, from some starting address to an ending address. It can be a snap dump or a dump ordered by the debug user as a last resort.

BLOCK SET MEMORY

This command provides the user with the capability of setting some block of memory to zeros or ones.

SEARCH A MEMORY AREA FOR A PATTERN

This command allows the user to determine the location of a particular value in mainstore. It can also allow a comparison with a mask for added flexibility in patterns.

PATCH AID

This command covers a wide range of possibilities. It can allow the debugger to control allocation within a space reserved for patches. It can help to insert jumps to form the patch. It can also provide an easy-to-use way to write the patch into the selected area (formatting, mnemonic translations, etc.).

CHECKPOINT/RESTART

This command pair allows the user to save the state of a program so that it can be restarted at that point at a latter time.

SYMBOLIC DEBUG

This command allows the use of a symbol table to address locations by symbolic names used at assembly or compile time.

APPENDIX B

EXTENSIONS TO DEBUGGING CAPABILITY

VALIDATION AND VERIFICATION

A possible approach to validation and verification which is being considered for the Trident project at NSWC/DL is described in a paper by Victor R. Basili and Robert E. Noonan.[1] This approach consists of monitoring test cases to determine a test set that causes all statements in the program to be executed. This kind of monitoring can be done on an emulator as in the Trident program; however, a few slight extensions to the AN/UYK-20 would allow this monitoring to occur in some approaches. The extensions would consist of the use of jump tracing, breakpoints, and possibly some additional special microcode to speed up monitoring. Although this might be accomplished via the external approach, it would be far more cumbersome than with the internal approach.

PERFORMANCE MONITORING

Although the addition of performance monitoring is similar to verification and validation, the aim is different. It includes features which determine where the program consumes most of its execution time, how much I/O activity there is and how long it takes to execute a segment of code. This can be performed with the external approach, but would be much more efficiently implemented by the internal and VM approaches.

REMOTE DIAGNOSTIC CAPABILITY

A possible extension of the hybrid and external approaches that has been mentioned is the use of the black box via a modem to allow a remotely located person to diagnose problems on the AN/UYK-20. This would enable the fleet to use land-based expertise via a phone link.

---

[1] Victor R. Basili and Robert E. Noonan, A Testing Tool for a Fire Control Environment, Department of Computer Science, University of Maryland.

APPENDIX C

VIRTUAL MACHINE APPROACH

One of the five approaches discussed in this report requires the construction of a virtual machine (VM). This appendix defines and describes this approach. There are three terms which require definition: virtual machine, virtual machine system, and virtual machine monitor (VMM).

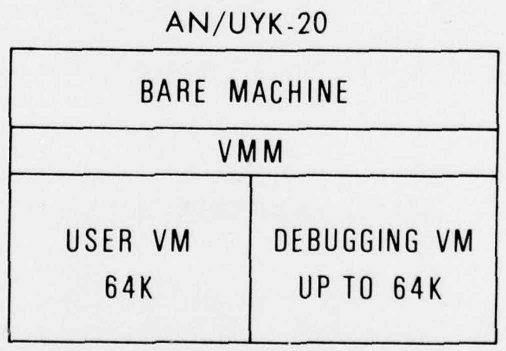
Virtual Machine - Performs the same function as the real computer

Virtual Machine System - A technique that handles one or more computers on a piece of hardware

Virtual Machine Monitor - Selects the computer to be placed into execution, then controls and monitors its operation

The word "computer" in these definitions is not used to denote a physical piece of hardware but rather as an abstraction embodying the capabilities of the hardware. In the case of the AN/UYK-20, we would use the word computer to describe a system with 16 general registers, 16-bit word length, and having a certain instruction repertoire and a 64K memory; however, this use of the word computer does not include the hardware used to implement the 16 registers, the instruction set, or the memory.

The VM approach to AN/UYK-20 debugging is to make the AN/UYK-20 a virtual machine system by allowing it to host a VMM. This is shown by the diagram below.



The VMM for the AN/UYK-20 is a combination of microcode and software which provides two VM's, one for debugging and one for user software.* Since there are finite resources in the AN/UYK-20, the VMM must control the sharing of these resources and ensure that the two VM's are transparent to each other. Thus, the user software is unaware that it is sharing the machine with a debugging package. Before describing the VMM in more detail, however, expansion of the AN/UYK-20 memory is discussed.

* The second virtual machine need not be used for debugging but could be a second independent process (i.e., user software, etc.).

The AN/UYK-20 allows a 64K memory, but the diagram above shows a 64K user VM and a debugging VM with up to 64K; therefore, additional memory must be added. This can be accomplished by developing a semiconductor memory board to double the memory of the AN/UYK-20. To access this added memory capacity, an extra address bit will be necessary. A bit can be obtained by using one bit of the status register (there are several unused bits) and adding one wire to the back plane. This wire would go from the status register to each memory board. As this wire would not be used in the normal AN/UYK-20 mode, it would be installed only on machines used for debugging. This change is not part of the virtualization process; rather, it is an extension to the AN/UYK-20. This extension not only doubles memory, but also creates two distinct address spaces in the AN/UYK-20 which can be used to protect VM memory spaces.

In general, a VMM is composed of a combination of hardware, software, and firmware. To minimize changes to the AN/UYK-20, the VMM discussed here is limited to software and firmware. This limits the range of performance possible with the VMM. Because of the sensitive nature of the AN/UYK-20 microcode, the use of firmware must be carefully considered during the design of the VMM. This will be a major trade-off and will have a decided effect on performance. The VMM for the AN/UYK-20 must handle three problem areas: (1) sensitive instructions, (2) I/O and interrupts, and (3) task swapping. These problems will be described in the following sections. The VMM uses CPU time on the real hardware; thus, the VMM creates overhead. This overhead could use 10 to 30 percent of the CPU time in the case of the AN/UYK-20, depending on the design of the VMM. This report presents only an outline of the work needed to create the VMM. In order to pursue this path, a further study is required to design a VMM for the AN/UYK-20.

SENSITIVE INSTRUCTIONS

There are certain instructions which can do critical things and can interfere with another user. These instructions must be changed in some way so that their effect is the same on the user program which executes them but does not affect the other VM's in the AN/UYK-20. All other instructions will execute in native mode, that is, without any modification to slow them down.

The Unary Control instruction (03 RR), the load and store address register instructions (54 and 55), and the I/O instruction (35) are the instructions which will require some changes.

The Unary Control has 16 subfunctions all of which could be critical in a virtual computer, executive return, store status register, real-time clock enables, etc. In particular, the status register manipulations must be changed so that the bit which addresses the expanded memory cannot be changed by a user program. This bit will be manipulated by the VMM only. Since both virtual computers can use the real-time clock, it may be necessary to emulate a real-time clock for each. Likewise, each will need a status register. Since only one virtual computer can run at a time, these functions can be performed by saving the status of a virtual computer when it is not running and restoring the status when it is to be run again. The two instructions which load or store address (page) registers may be important to the VMM. The page registers can be used to protect memory spaces. AN/UYK-20 page registers cannot be disabled; they always address 64K pages of memory. However, pages (1K blocks) of physical memory may be protected if no page register points to them. To protect a page, two (or more) page registers are pointed to the same memory page which is in use; thus, several logical addresses will point to the same physical memory addresses, while other physical memory addresses are not addressable. If the VMM is to use this technique, the load and store address register instructions will require protective measures.

The sensitive instructions must be protected; since the AN/UYK-20 does not have a supervisor state to do this, another method must be employed. One approach is to change the ECW's for these instructions to point to a new microroutine that performs the checking required and prevents the user from interfering with the other VM's in the system. These microroutines are part of the VMM.

I/O AND INTERRUPTS

The I/O creates some severe problems. I/O uses the same page registers to access memory as the CPU; therefore, I/O in one virtual computer has difficulty running when another virtual computer is in execution. A possible solution to this is not to allow I/O from one virtual computer to run when it is not executing. This, however, is inefficient in that we would like to allow concurrent I/O operation. In order to allow concurrent I/O operation, the VMM must be able to associate an address produced by the I/O channel with a physical memory address. The physical address not only includes the 16 bits normally used, but also the status register bit which is used to address the additional memory. Thus, in this scheme, each I/O transfer will have to trap to the VMM where a memory mapping can occur. If the hardware of the AN/UYK-20 can be changed slightly, this mapping can be in hardware; otherwise, it creates a costly overhead on each transfer.

There is a possible solution to this for the debugging case. Only one I/O device is needed in the debugging machine; therefore, all I/O except on one channel goes to the user's VM. If the VMM is constructed with this restricted case in mind, an I/O transfer will always go to the user memory space. When the debugging VM is running, it will use some of the user memory (this will be transparent to the user) as a buffer for its own I/O transfers. This will reduce the overhead required on each I/O transfer.

The interrupt system must also be changed so that interrupts go first to the VMM. The VMM must route the interrupts to the correct VM. If the VM receiving the interrupt is not running, the interrupt must be saved until it is in execution.

TASK SWAPPING

The last topic of discussion is task swapping: How does one virtual computer stop and another begin execution? This can be performed by either time slice scheduling or interrupts. Another feature of task swapping is that the VMM must have a save area in which the status of each virtual computer is contained plus a set of constructs for affecting the swapping of a virtual computer from the executing to the nonexecuting state. These constructs will be put into microcode so that only the VMM can execute them. In addition to the task swapping constructs, the VMM will need a spy construct that is used to examine the state of the operational software and a breakpoint OP code (diagnostic return could be used). This OP code will be used to schedule the debugging VM.

CONCLUSION

The above discussion indicates that it should be possible to virtualize the AN/UYK-20. However, it may be easier to virtualize the AYK-14 and use it to debug the AN/UYK-20 software.

DISTRIBUTION

Naval Electronics Sytems Command
Washington, DC 20360
Attn: CAPT Hager
ELEX-570

Johns Hopkins University
Applied Physics Laboratory
Johns Hopkins Road
Laurel, MD 20810
Attn: Dr. Mars Gralia

Defense Documentation Center Cameron Station Alexandria, VA 22314	(12)
Defense Printing Service Washington Navy Yard Washington, DC 20374	
Library of Congress Washington, DC 20540 Attn: Gift and Exchange Division	(4)

Defense Advanced
Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209
Attn: William Carlson

U.S. Naval Electronic
Systems Command
Washington, DC 20360
Attn: John Machado (Code 330)
R. Fratila (Code 330)

Local:

CN-20 (John Straub)
CK
CK-60
CK-70
CK-74 (15)
DX-21 (2)
DX-222 (6)
DX-43 (Green)
DX-40